CS 3EA3 - Lecture Notes
Heuristic Programming

Danish Nadeem

March 31st, 2017

Recall "Deleting a conjunct"

If R is of the form P ∧ B with P -> easily truthified

B -> "hard"

Then consider P as *invariant* and B as *loop guard*. → Initialize P

;do ¬B ->??? od

{P ∧ B ie. R}

For the following examples refer to the provided checklist on the theorem sheet

Also given here:

Program Construction

Heuristic "Programming is a *goal-oriented* activity":

1. Formalise '*Givens*' and '*Requireds*' of the problem.
2. Obtain an invariant P and initalise the variables to make it true.
3. Bridge from invariant to post-condition: solve for B in $P \wedge \neg B \Rightarrow R$
4. If $\neg B$ holds then we're done, otherwise we construct a loop to obtain it.
5. Solve for a "bound function" bf in $P \wedge B \Rightarrow bf > 0$.
6. Make progress towards termination: find a program S that *decreases* the bound.
7. Refine program S so that it *maintains* the invariant!

$\{G\}$

intialisation

{invariant P; bound bf}

;**do** $B \rightarrow \{P \wedge B \wedge bf = C\}\ S\ \{P \wedge bf < C\}$ **od**

$\{R\}$

Note: the following program appeared in lecture one

R: x ≤ y "x is atmost y"

Is the same as:

- x ≤ y ∧ true (idea of "multiply by 1")
- true ∧ x ≤ y

bf: x > y

x-y>0

Program:

do (x>y) → x,y := y,x od *this will do one iteration

Side note: an IF statement was not used in the following program because if one guard is not true then the program will simply crash or abort.

```
if
  ▌G1 ->
  .
  .
  ▌Gn ->
fi
```

*considering the if statement only consists of conditions and not an else statement: there is no guarantee that at least one guard will be true.

Example 1:

Note: The following program appeared in lecture 1. Similar to 3 elements de-arrangements.

R: $a \le b \le c \le d$

- true $\wedge$ $a \le b \wedge b \le c \wedge c \le d$ "multiplying by 'one' concept"
- P $\wedge$ B

B → negation of whole thing as loop guard: (recall: De Morgan law)

$\neg(a \le b \wedge b \le \wedge\ c \le d) = a>b \vee b>c \vee c>d$

Bound function: the number of out of order elements

do

▌a>b → a,b:=b,a

▌b>c → b,c:=c,b

▌c>d → c,d:=d,c

od

with this we have to check if invariant holds at every position; but true will always be true

Example 2:

R: $q=A \div B$ ($\wedge r = A \bmod B$)

- $A = q \cdot B + r \ \wedge 0 \le r < B$ → $0 \le r \ \wedge \le r < B$

(P) (B)

G: $0 < B$

we know r is at least b so we can decrease: r by B

recall the notion of changing it to a domain you understand

***Ch15 - quiz sheet 6/7*

Refer to end of these notes for link

Find values for q and r that make R true.

If it's hard, that will give you loop guard

Loop guard is negation of B. $r<B \rightarrow \neg(r<B) = r \ge B$

If we take q=0, then A=r. so given is now:

G: $0<B \wedge 0 \le A$

Bf: $P \wedge B \le r$

={Weakening}

$B \le r$

⇒{given and transitivity}

$0 < r$

Program:

q,r=0,A

do $B \le r$ → q,r:=E,r-B od call it E for expression we don't know

if $P \wedge (B \le r)$ then

$P[q,r:=E,r\text{-}B]$

={defn of P and textual substitution}

$A=E \cdot B+ r\text{-}B \ \wedge 0 \le r\text{-}B$

={given $B \le r$ and P}

$q \cdot B + r = E \cdot B + r\text{-}B$

={arithmetic}

$E=q+1$

now we can write in code: do $B \le r$ → q,r:=q+1,r-B od

Theorem sheet: http://www.cas.mcmaster.ca/~alhassm/ThmList.pdf

Links referring to sheet 6 and 7:
http://www.cas.mcmaster.ca/~alhassm/Sheet6.pdf
http://www.cas.mcmaster.ca/~alhassm/Sheet7.pdf